

METHOD/DEVICE FOR COMPILING AND METHOD/DEVICE FOR CALCULATING METHOD ACTIVITY DEGREE

Patent number: JP2000222221
Publication date: 2000-08-11
Inventor: OGASAWARA TAKESHI
Applicant: INTERNATL BUSINESS MACH CORP < IBM>
Classification:
- international: G06F9/45; G06F9/42; G06F11/34
- european:
Application number: JP19990022030 19990129
Priority number(s):

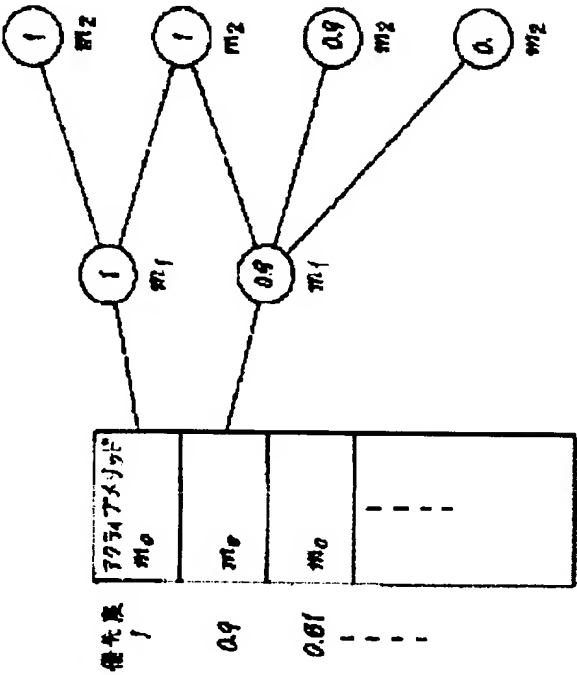
Also published as:
US6671877 (B1)

Report a data error here

Abstract of JP2000222221

PROBLEM TO BE SOLVED: To provide a method for calculating a method activity degree for effectively selecting a method for destructing a code.

SOLUTION: In compiling, a calling map concerning method calling which can be generated in the method is prepared and stored in a storing device, and a code for recording method calling generated actually is generated and stored in the storing device. In recovering the code in a state like this, all the threads are temporarily stopped to calculate activity degree expressing high or low possibility that some method is to be executed. Through the use of the calling map and information concerning method calling generated actually concerning a first method corresponding to a stuck frame, a second method of high possibility of being called from the first method is specified and stored in the storing device and priority according to the number of stages from the highest part of the stuck of the stuck frame corresponding to the first method is operated to the activity degree of the second method to update the activity degree of the second method. It is suitable to repeat processing like this. After then, the code of a method of a low activity degree is destructed.



(19) 日本国特許庁 (J P)

(12) 公開特許公報 (A)

(11) 特許出願公開番号
特開2000-22221
(P2000-22221A)

(43) 公開日 平成12年8月11日 (2000.8.11)

(51) Int.Cl. ⁷	識別記号	F I	テマコード* (参考)
G 0 6 F 9/45		C 0 6 F 9/44	3 2 2 A 5 B 0 3 3
9/42	3 3 0	9/42	3 3 0 A 5 B 0 4 2
11/34		11/34	S 5 B 0 8 1

審査請求 有 請求項の数9 O L (全 8 頁)

(21) 出願番号 特願平11-22030

(22) 出願日 平成11年1月29日 (1999.1.29)

(71) 出願人 390009531
インターナショナル・ビジネス・マシー
ズ・コーポレーション
INTERNATIONAL BUSIN
ESS MASCHINES CORPO
RATION
アメリカ合衆国10504、ニューヨーク州
アーモンク (番地なし)
(74) 代理人 100086243
弁理士 坂口 博 (外4名)

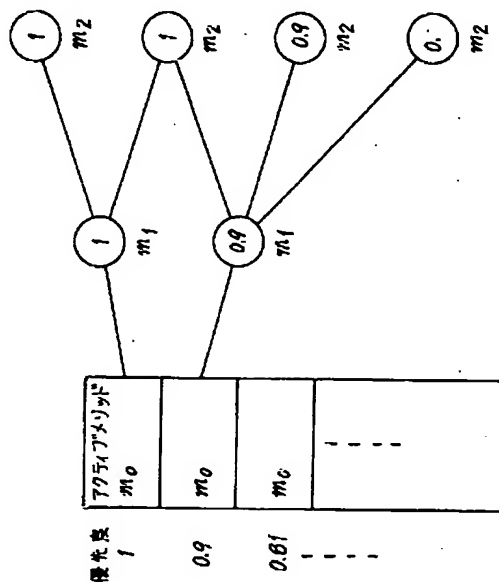
最終頁に続く

(54) 【発明の名称】 コンパイル方法および装置、並びにメソッド活動度計算方法および装置

(57) 【要約】

【課題】 コード破棄するメソッドを効果的に選択するためにメソッド活動度を計算する方法を提供する。

【解決の手段】 コンパイル時に、メソッド中で生じ得るメソッド呼び出しに関する呼び出しマップを作成し且つ記憶装置に格納し、実際に生じたメソッド呼び出しを記録するコードを生成し且つ記憶装置に格納する。このような状態において、コードの回収する際には、全スレッドを一時停止し、あるメソッドが実行される可能性の高低を表す活動度を計算する。この時、スタックフレームに対応する第1メソッドについての、呼び出しマップと実際に生じたメソッド呼び出しに関する情報とを用いて、第1メソッドから呼ばれる可能性が高い第2メソッドを特定し且つ記憶装置に格納し、第1メソッドに対応するスタックフレームのスタックの最上部からの段数に応じた優先度を第2メソッドの活動度に作用することにより第2メソッドの活動度を更新する。好適には、このような処理を繰り返す。その後活動度が低いメソッドのコードを破棄する。



edコードを実行するのが一般的である。コンパイルの最小単位は、メソッドと呼ばれるサブルーチンである。実行頻度の高いコードを機械語に変換することにより、Javaバイトコードのマルチプラットフォーム特性を残しつつ、機械語コードの性能を引き出している。

【0004】メソッドのJITedコードは、プログラムが動作するCPUに適したコードになっているという点で、Cコンパイラなどが生成する最適化コードと同等である。一般に、現在普及しているCPUでは、サブルーチンが呼ばれると、サブルーチンは、自分が使用するローカル変数を格納するための領域（フレームと呼ぶ）をスタック状に形成する。図1は、サブルーチンAがサブルーチンBを、サブルーチンBがサブルーチンCを呼び出したときのスタックの様子である。この図において、スタックは下から上に伸びる。各領域は、サブルーチンのそれぞれのフレームを表わす。各サブルーチンから戻る際、フレームは除去される。Javaでは、スレッドと呼ばれる実行単位毎にCPU資源が割り振られ、1つのスレッドは固有のスタック（スレッドスタックと呼ぶ）を持つ。JITedコードは、スレッドスタック上に上述したようなフレーム（以下、JITedフレームと呼ぶ）を作る。

【0005】メソッドは、最初ほとんどの場合インタプリタで実行されるが、実行頻度あるいは実行時間においてある水準以上実行されると判断されると、JITコンパイラによりコンパイルされる。JITedコードは一度生成されると、メモリ中に管理される。それらを安全に捨てるができるということJavaシステムが保証する時期は、Javaシステムがガベージコレクション（以下、GC）によってクラスを破棄する（以下、クラスアンロード）ときである。

【0006】JITedコードを含むJITコンパイラが使用するメモリに制限がある、あるいは制限を設けるとき、クラスアンロードだけでは十分ではない場合がある。例えば、ハードディスクによる仮想記憶を持たない、ネットワークコンピューティング（Network Computing）マシン（以下、NCマシン）のような、シンクライアント（thin client）上で動作するデスクトップ環境では、メモリ制限は現実的な問題である。そのようなデスクトップ環境では、メール、ワープロ、スケジューラ等、複数のJavaアプリケーションが同時に起動され、数千種類のメソッドが呼ばれうるが、かなりの割合のメソッドが各スレッドスタック中に存在しない（アクティブでない）。クラスアンロードは、Javaシステム内に完全に使用される可能性のないクラスおよびそのメソッドしか破棄しないため、そうしたアクティブでないメソッドのJITedコードを破棄しない。そこでJITコンパイラはJITedコード破棄メカニズムを持っている。

【0007】実行中のスレッドの現在のコンテキスト

（プログラムカウンタ（program counter）やスタックポインタ（stack pointer）などCPU資源のコピー）あるいはそのスレッドスタック中に保持されるJITedコードのフレームには、そのスレッドでのアクティブなメソッドが記録されている。アクティブなメソッドのJITedコードが破棄されると、スレッドが実行できない。したがって、破棄するのは、非アクティブなJITedコードである。JITedコード破棄メカニズムにおいて最も簡単な破棄の仕方は、アクティブでないJITedコードをすべて破棄する方法である。しかし、この方法では必要以上にJITedコードを破棄してしまい、無駄である。メソッドの最適化コンパイルには時間がかかるため、「すぐに実行されるメソッドを破棄しない」という方針の下で選択的に破棄すべきである。的確な選択の効果の例として、JITedコード破棄が今行われ、すぐに実行されるメソッドAと、10分後に実行されるメソッドBがあったとする。もしAを破棄すると、今破棄したAのJITedコードのメモリ量と、さらにAのコンパイルに必要なワークのメモリ量とを、直ちに確保しなければならぬ。一方、Bを破棄すれば、10分間はBのJITedコードのメモリ量は、利用可能メモリとなる。実際には必要なフリーメモリ量分の複数のJITedコードを破棄する。理想的な破棄の方針は、今後使用されないJITedコードを破棄することであるが、このような情報は普通得られない。類推し、しばらく呼ばれそうにない順にJITedコードを破棄するのが現実的である。以下、これをJITedコード破棄方針と呼ぶ。

【0008】通常考えられるメソッドのプロファイルに、実行頻度（何回呼ばれたか）と実行時間（プログラムカウンタがそのメソッドを指していた時間）がある。実行頻度の収集は、メソッドが呼ばれる毎にそのメソッドのカウントを増加させて行なう。メソッドがJITコンパイラによるコンパイルされると、それまでインタプリタで測定されていた実行頻度の更新は終了し、したがってインタプリタからはJITedコードに対する実行頻度情報は得られない。JITedコードで実行頻度情報を得るには、各メソッドのJITedコードのプロローグで実行回数を数える必要がある。例えば、x86プラットフォームでは、inc dword ptr [カウンタアドレス] 命令を追加する。実行時間の収集は、一定時間毎に割り込み（典型的にはタイマ割り込み、プロセッサによっては実行命令数割り込み）をかけ、プログラムカウンタ（以下、pc）を収集して行なう。

【0009】JITedコード破棄方針として、これら従来のプロファイルを使用すると問題が生じる。まず、実行頻度について、各メソッドのJITedコード実行の度にメモリを参照してカウントするのは、メモリが枯渇しない状態へのオーバーヘッドになり、問題である。さらに、頻度が少ないが直ちに呼ばれるメソッドを誤って

破棄してしまう恐れがある。知りたいのは、呼ばれずにメモリを占めているJITedコードであるから、目的に合わない。また実行時間についても、割り込みと収集したpcの管理（すぐにメソッドに対応づけるなら対応づけのコスト、対応づけなければその保存）は、メモリが枯渇しない状態へのオーバヘッドになり問題である。さらに実行頻度収集の場合と同様、実行時間が短いが直ちに呼ばれるメソッドを誤って破棄してしまう恐れがある。

【0010】

【発明が解決しようとする課題】上述のように、従来のプロファイル手法をJITedコード破棄方針とするのは外的で、不適切なJITedコード破棄が起きる可能性がある。本発明の目的は、コード破棄するメソッドを効果的に選択するために、メソッドがすぐに実行されやすいことを表わすメソッド活動度を計算する方法を提供することである。

【0011】

【課題を解決するための手段】本発明は上記課題を解決するため、コンパイル時に、メソッド中で生じ得るメソッド呼び出しに関する呼び出しマップを作成し且つ記憶装置に格納し、実際に生じたメソッド呼び出しを記録するコードを生成し且つ記憶装置に格納する。この呼び出しマップは、好適には、メソッド中のあるアドレスに対し、当該アドレス以降に生じ得るメソッド呼び出しを出力するように構成される。但し、メソッドごとにメソッド呼び出しを出力するようにしてもよい。コンパイルが終了し且つ生成されたコードが実行されると、メソッド中で生じ得るメソッド呼び出しに関する呼び出しマップと実際に生じたメソッド呼び出しに関する情報とが各メソッドごとに記録されるようになる。

【0012】このような状態において、コードを回収する際には、全スレッドを一時停止し、あるメソッドが実行される可能性の高低を表す活動度を計算する。この時、スタックフレームに対応する第1メソッドについての、呼び出しマップと実際に生じたメソッド呼び出しに関する情報とを用いて、第1メソッドから呼ばれる可能性が高い第2メソッドを特定し且つ記憶装置に格納し、第1メソッドに対応するスタックフレームのスタックの最上部からの段数に応じた優先度を第2メソッドの活動度に作用することにより第2メソッドの活動度を更新する。好適には、このような処理を繰り返す。すなわち、第2メソッドについての、呼び出しマップと実際に生じたメソッド呼び出しに関する情報とを用いて、第2メソッドから呼ばれる可能性が高い第3メソッドを特定し且つ記憶装置に格納し、第1メソッドに対応するスタックフレームのスタックの最上部からの段数に応じた優先度を第3メソッドの活動度に作用することにより第3メソッドの活動度を更新する。このようにして求められたメソッドの活動度が低いもののコードを破棄するとより効率

的なコード破棄が行える。

【0013】なお、好適には、呼び出しマップは、メソッド中のあるアドレスに対し、当該アドレス以降に生じ得るメソッド呼び出しを出力し、第2メソッドの特定は、第1メソッドの実行再開アドレス以降に生じ得るメソッド呼び出しを用いて行われる。

【0014】本発明では、非アクティブメソッドについて、現在のコンテキスト、プログラムの構造、実行時情報を利用して、すぐに呼ばれる度合い（活動度）を求める。ここでいう、現在のコンテキストの利用とは、スタック上位（トップに近い）にあるメソッドほど早く実行されそうだと判断できることである。プログラムの構造の利用とは、コントロールフロー上、一時停止しているメソッドから今後呼ばれそうなメソッドを知ることである。実行時情報の利用とは、前記呼ばれそうなメソッドでも（インタプリタを含めた）実行時に一度も呼ばれていなければ、判断対象から除外できるということである。早く呼ばれそうなメソッドの活動度が、その呼ばれる早さに応じて上げられる。

【0015】このように活動度は、メソッドの再実行時刻が早いほど、その値が大きい。これは、上述したJITedコード破棄方針をよく模倣する点で優れている。先ほどの例でいえば、仮にAがスタック上位、Bがスタック下位からそれぞれ呼ばれるメソッドであれば、本発明のアルゴリズムによってAの活動度はBの活動度より高くなり、Bより先にAを選んで破棄することとはなくなる。

【0016】あるスレッドでJITコンパイラのメモリ要求が満たせなくなったとき、そのメソッドは自分以外のスレッドをすべて一時停止し、アクティブメソッドを見つける。本発明によるメソッド活動度計算方法により、各メソッドの活動度を決定する。その後、このような活動度に基づいて、非アクティブなメソッドの一部あるいは全部のJITedコードを破棄する。一時停止したスレッドをすべて再開し、メモリ要求を再度行なう。これをメモリ要求が満たされるまで行なう。最終的にメモリ要求が満たされない場合、許されるなら、コンパイラの最適化レベルを下げて再度コンパイルに挑戦する。再度のコンパイルが許されない場合、あるいは最低の最適化レベルでもコンパイルできなかった場合は、このメソッドをコンパイルしない。

【0017】メソッド活動度を計算するための作業コストを低減するために、メソッドの活動度を、これらがさらに呼び出すメソッドの活動度に作用させる動作の回数に上限を設けてもよい。

【0018】

【発明の実施の形態】本発明は、JITedコード破棄方針を模倣するために、メソッドに活動度と呼ばれる優先順位を与える方法である。JITedコード破棄方針とは、「しばらく呼ばれそうにない順にJITedコー

ドを破棄する」ことである。低い活動度のメソッドほど、しばらく呼ばれないと予想できる。しばらく呼ばれないメソッドのJITedコードを破棄すれば、それらによって使われていたメモリ量のフリーメモリは長い間活用できるはずである。

【0019】以下、メソッドmの活動度A(m)の計算手順を示す。各メソッドの活動度の初期値を0とする。まず、JITコンパイラによるコンパイルおよび実行時における動作を、図2のフローチャートを参照して説明する。ステップS201で開始し、ステップS202で、メソッドmに関して、JITedコード中のアドレスaddrと、このアドレスからメソッドの実行が再開したときに呼ばれる(あるいは可能性がある)メソッド集合とを対応させるための、アドレス-呼び出しマップMAPA-I(m)を作る。アドレス-呼び出しマップは、addrを含む基本ブロックBB(basic block)と、このBBからコントロールフロー上で到達する前記メソッド呼び出しとを対応づけるものである。このアドレス-呼び出しマップは、あるアドレスaddrが与えられたときに、このaddrから到達する全メソッド呼び出しを返す。この全メソッド呼び出しを呼び出し集合SETA-I(m, addr)と呼ぶ。

【0020】アドレス-呼び出しマップの計算をするためには、コントロールフローを解析し、基本ブロックBB毎に、BBが到達する可能性のあるメソッド呼び出しのアドレスを集める。それから、BBの先頭アドレスi_addrとリーチされるメソッド呼び出しアドレスi_callの関係データ(例えば、(i_addr, i_call)をソートした表)をコンパイルコードにつける。呼び出し集合は、addrを含むBBに対応するすべてのi_callである。図3は、メソッドEのJITedコードを示す線図である。メソッドEは、BB1ないしBB5から成る。この図において、アドレス115において、メソッドFを呼び出している。したがって、呼び出し集合は、addr=100~115はFを返し、addr=116~120は何も返さない。

【0021】アドレス-呼び出しマップの計算コストを大幅に削減するには、正確でないが、メソッドmのアドレス-呼び出しマップを、mに含まれるすべてのメソッド呼び出しの集合としてもよい。このとき呼び出し集合もmに含まれるすべてのメソッド呼び出しの集合である。

【0022】JITedコード中にあるメソッド呼び出しのためのcall命令は、最適化のために、なるべくダイレクトコール(direct call)になるように処理される。例えば、あるメソッド呼び出しをコンパイルする際に呼び出し先のメソッドがまだコンパイルされていない場合、コンパイラはダイレクトコール命令を生成できない。このようなメソッド呼び出しが発生すると、バックパッチコードにジャンプして、JITコンパイラによ

って生成されたターゲットアドレスを次回のメソッド呼び出しから用いることができるように設定する。このような処理は従来から行われてきた。本発明では、メソッド呼び出しを最初に行う場合には、ターゲットアドレスがまだ生成されていない場合でも生成されている場合でも必ず、バックパッチコードにジャンプし、ターゲットアドレスを記録するようにする(S203のコードを用いる)。2回目以降のメソッド呼び出しでは、バックパッチコードにジャンプする必要はない。

【0023】ステップS203で、メソッド呼び出しのバックパッチコード中に、各呼び出しコードによって実際に呼び出しが起きたことを記録するコードを生成する。この記録を有効呼び出し集合SETEI(m)と呼ぶ。この有効呼び出し集合は、呼び出しアドレスとターゲットアドレスのペア(ei_call, ei_target)の集合である。ei_targetの初期値はNULLである。ei_targetがNULLでない要素が意味のある要素である。

【0024】ステップS204で、コンパイラは、メソッドをコンパイルする際に、全てのターゲットアドレスについての表(ei_call, ei_target)を作り、コンパイルコードに付ける。例えば、表1のようなものである。

【表1】

ei_call	ei_target
xxxx	null
yyyy	null
zzzz	null
.	.

この表が有効呼び出し集合となる。コンパイル時の処理は以上である。

【0025】JITedコードが実行され、S203で生成されたコードが実行されると、呼び出しを行なったメソッドの有効呼び出し集合の対応するエントリ(ei_call, ei_target)のei_targetが更新される。ここで、有効呼び出し集合においてei_targetに実際のターゲットアドレスが入っている呼び出しのみが、実際に呼び出しを行なう有効呼び出しである。ステップS206で処理を終了する。なお、表1は、例えば表2のようになる。

【表2】

Tコンパイラ56を含む。JavaVM52は、図示しないガベージコレクタやスタクトレーサをさらに含む。なお、クライアント・コンピュータ5は、通常のコンピュータの他、メモリの大きさが小さかったり、ハードディスク等の補助記憶装置を含まないような、いわゆるネットワーク・コンピュータや情報家電の場合もある。

【0040】サーバ・コンピュータ1では、Javaソースコード10は、Javaコンパイラ12によりコンパイルされる。このコンパイルの結果が、バイトコード14である。このバイトコード14は、ネットワーク3を介してクライアント・コンピュータ5に送信される。バイトコード14は、クライアント・コンピュータ5内のWWWブラウザ（World Wide Web Browser）などに設けられたJava仮想マシン（Java VM）52にとってネイティブ・コードであり、実際ハードウェア55のCPUにて実行する場合には、Javaインタプリタ54や、JavaJITコンパイラ56を用いる。インタプリタ54は、実行時にバイトコード14をデコードし、命令ごとに用意される処理ルーチン呼び出して実行する。一方、JITコンパイラ56は、バイトコードを事前にあるいは実行する直前にコンパイラを用いてマシン・コード58に変換してそれをCPUで実行する。

【0041】

【発明の効果】本発明によれば、コード破棄するメソッドを効果的に選択することができるため、同一メソッドの再コンパイル回数が減少し、コンパイルオーバーヘッドを減少させることができる。

【図面の簡単な説明】

【図1】 サブルーチンAがサブルーチンBを、サブルーチンBがサブルーチンCを呼び出したときのスタックを示す線図である。

【図2】 本発明によるメソッド活動度計算方法のJITコンパイルおよび実行時における動作を説明するフローチャートである。

【図3】 メソッドEのJITedコードを示す線図である

【図4】 本発明によるメソッド活動度計算方法のJITedコードの回収プロセス中における動作を説明するフローチャートである。

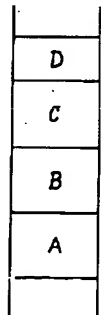
【図5】 ターゲットメソッドへの活動度の割り当てを説明する線図である。

【図6】 本発明における装置構成の一例を示す図である。

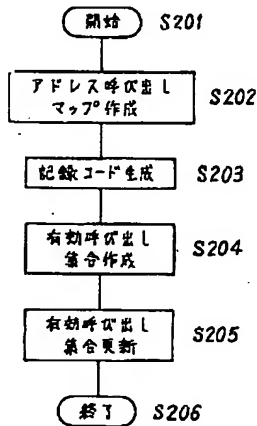
【符号の説明】

- 1 サーバ・コンピュータ
- 3 ネットワーク
- 5 クライアント・コンピュータ
- 10 Javaソースコード
- 12 Javaコンパイラ
- 14 バイトコード
- 52 JavaVM
- 54 Javaインタプリタ
- 56 Java JITコンパイラ
- 58 マシンコード
- 60 ガベージ・コレクタ
- 53 OS
- 55 ハードウェア（CPU及びメモリを含む）

【図1】



【図2】



【図5】

